Portable Multibody Simulation Using MPI

Ruaidhrí O'Connor*
Department of Civil and Environmental Engineering, Room 1-245
Massachusetts Institute of Technology, Cambridge, MA 02139
Phone: x3-2708 Email: roryoc@mit.edu

Abstract

This paper describes two components of a multibody dynamics system implemented using the *message passing interface* library, MPI. A simple *force decomposition* scheme (FD) is used to parallelize the application by distributing the *contact detection* task over available processors. Using the FD scheme we obtain a near optimal linear speedup with increasing number of processors on dedicated parallel hardware. We introduce a new object representation scheme referred to as the *discrete function representation*, DFR. The DFR scheme describes a data structure for *non-convex* 3D geometries and we use it to perform efficient contact detection calculations local to a processor. Timing results are presented for sample runs of the system containing up to 10,000 bodies (on an IBM SP2, a Thinking Machines CM-5 and a small network of Sun workstations).

Overview

In recent years there have been several advances in the development of efficient parallel N-body applications that allow researchers to consider systems containing $10^6 - 10^8$ bodies (notable examples can be found in [4, 2]). For systems of this size it is possible to observe macroscopic characteristics of material behavior starting from the atomic level. However, in these very large systems the bodies are relatively simple. Each is represented as a point charge or mass. The complexity in determining the evolution of the system resides in reducing the long range *all-to-all* coupling between bodies. This is accomplished using clustering techniques and/or multipole expansions. In contrast, the sort of multibody problems we are interested in contain objects which have a considerably more detailed *local* representation. These multibody systems arise in the modeling of particulate materials such as sands. Each grain of the material is represented as a unique object with a specific geometry and physical state. While these characteristics only have *short range* influence on the system, the overhead associated with calculating the details of geometric intersection can make a system, as small as 10,000 grains, too costly to consider over the time scales of interest. It is of more value to us to run a problem of this size for large numbers of times steps, as opposed to very large numbers of simple objects for much shorter periods. As a consequence, we have directed our attention to reducing the time complexity of the problem by developing efficient serial and parallel algorithms briefly described in the following sections.

Discrete Function Representation (DFR)

The strategy underlying the DFR scheme is to store the surface geometry of an object in the form of a 3D lookup table or *stencil*. The surface geometry is first tessellated on a regular grid of cells or *voxels*. As the tessellation proceeds, a topologically equivalent 3D array of memory (the *stencil*) is generated, where each voxel that straddles or lies inside the bounding hull of the surface is minimally stored as a single word of memory, serving as a discrete analog of the actual surface.

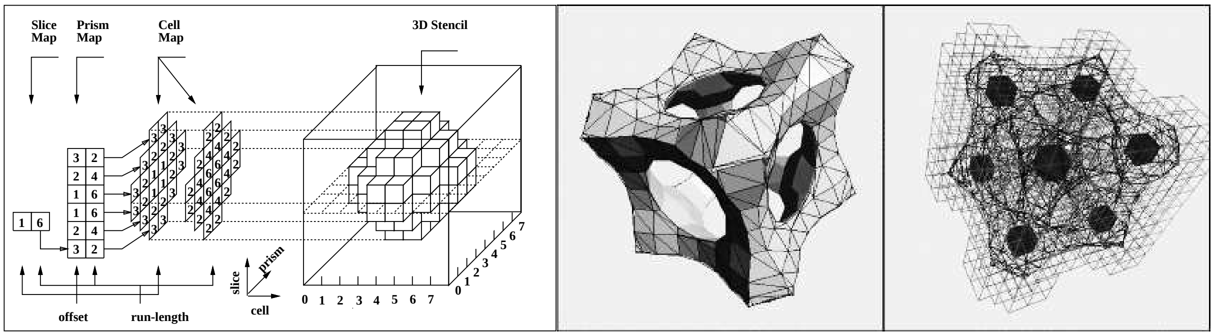

Figure 1: a) Internal Structure of the DFR. b) Example DFR Object - Porous Grain

*Research Advisor: John R. Williams (john@iesl.mit.edu). Research funded in part by IESL Corporate Affiliates.
CM-5 resources at MIT LCS made available through Project SCOUT, sponsored under ARPA contract #MDA972-92-J-1032.
SP2 resources at MIT RLE made available through participation in IBM's Shared University Research Program (SUR).

The DFR data structure is stored as a ragged 3D array of cell indices called the **3D stencil**, and a set of ragged arrays containing *offset*, *run-length* (ORL) pairs to describe the **stencil** layout. This structure is shown in Figure 1.a with an example non-convex object geometry in Figure 1.b. The **3D stencil** is used to store indices into a table containing the tessellated surface description (a table of vertex vectors) which is stored separately (not shown). Aligning the 3D stencil layout with the local coordinate system of the object, any region of the surface can be directly accessed through a *single* scaling and translation operation. This property allows us to test if a point lies inside a discrete convex hull of the object, in constant time. To test all m sample points of one surface against the DFR of another surface then takes $O(m)$ operations. This is compared with standard methods which take between $O(mlog(m))$ and $O(m^2)$ or worse for non-convex surfaces. In practice, because the storage scheme is so structured, the actual number of points that need to be tested will be proportional to the region of intersection, which averages closer to $\sqrt{m}$ operations.

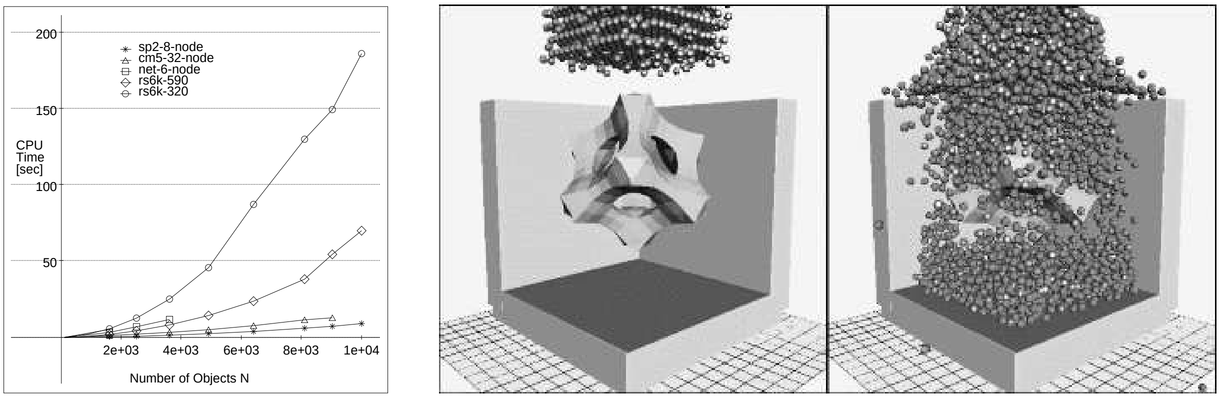

Figure 2: a) CPU Time Per Iteration For Different Size Systems. b) Frames from Simulation Video.

Force Decomposition

The parallelization focused on distributing the two most expensive tasks in the simulation, collectively referred to as *contact detection*. The first major task is to sort and search for neighboring objects with which contact must be checked. The second task is calculating the details of the contact, geometric intersection and force resolution. We have found that these components account for almost 90% of the computation involved in a large multibody simulation. We examined several strategies for distributing the sorting operation, [1, 3], but found that serially sorting the entire dataset on every node was slightly faster than a parallel *mergesort* for this size of dataset. This factor led us to develop a simple *force decomposition scheme* where a copy of the data set is placed on each processor and all trivial local calculations duplicated in an attempt to minimize communications and maintain a reasonably synchronous execution. For a system containing N objects and P processors, each processor is made responsible for calculating the contact resolution details for each unique subset of N/P objects. Contact force components are calculated using the DFR scheme and *combined* by folding in $log(P)$ phases, yielding an updated state for the entire system on each processor. All interprocessor communication was performed using MPICH[1]. Using the library allowed us to prototype in a network environment avoiding the need to vie for resources on parallel machines during development.

The performance of the schemes is graphed in Figure 2.a and video frames from a reference 3D simulation containing 10,000 objects are shown in Figure 2.b. The performance results are for 2 single processor IBM RS/6000s (320H and 590), a network of 6 Sun workstations, 32 nodes of a CM-5 (without VUs) and an 8 node IBM SP2.

References

[1] Selim G. Akl. *Parallel Sorting Algorithms.* Academic Press Inc., HBJ Publishers, New York, 1985.

[2] D. Beazley and P. Lomdahl Large-scale molecular dynamics on mpps. *SIAM News*, 28(2), 1995.

[3] Guy E. Blelloch, Charles E. Lieserson, Bruce M. Maggs, C. Greg Plaxton, Stephen J. Smith, and Marco Zagha. A comparison of sorting algorithms for the connection machine cm-2. *Proceedings ACM Symposium on Parallel Algorithms and Architectures*, 1991.

[4] Michael S. Warren and John K. Salmon. A parallel hashed oct-tree n-body algorithm. In *Proceedings of Supercomputing '93*, Los Alamitos, IEEE Comp. Soc., 1993.

[1] MPICH was developed at Argonne National Laboratory and Mississippi State University as a portable implementation of the emerging **MPI** (Message Passing Interface) standard.